Procedural programming Code

Petr Svarny

Programming before procedures

Machine code (online)

Direct CPU control, relevant for modern security analysis or reverse engineering

```
0:      b8 05 00 00 00          mov    eax,0x5
5:      c3                      ret
```

Assembler (online)

Assembler summary, Tutorial

Machine specific low-level language

```
mov     eax, [x]
sub     eax, '0'
mov     ebx, [y]
sub     ebx, '0'
add     eax, ebx
add     eax, '0'
```

Basic (original) (online)

```
1000 REM Fibonacci Sequence Project

1010 REM Quite BASIC Math Project

1020 REM ------------------------

2010 CLS

2020 REM The array F holds the Fibonacci numbers

2030 ARRAY F

2040 LET F[0] = 0

2050 LET F[1] = 1

2060 LET N = 1

2070 REM Compute the next Fibbonacci number
```

Procedural programming

Procedural programming paradigm

- Procedural programming is a programming paradigm, derived from imperative programming, based on the concept of the procedure call. Procedures simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or itself.

What makes it stand out?

- Grouping of instructions into procedures

Procedural programming Languages

https://curlie.org/Computers/Programming/Languages/Procedural

How to show a language?

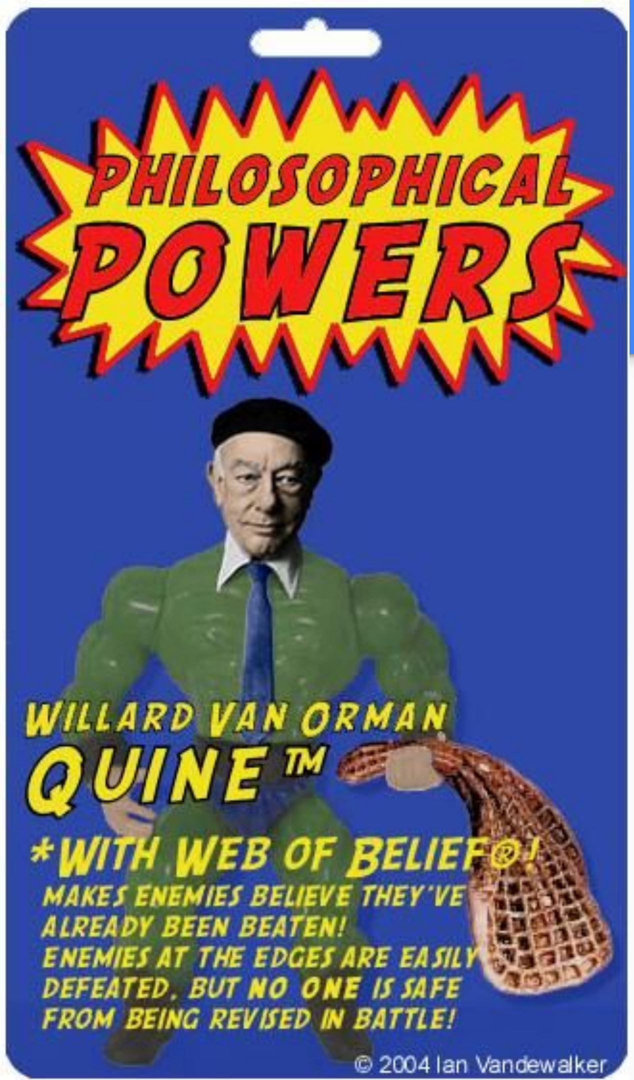
"Hello world"s

- output Hello world

"Quine"s

- output itself

https://www.wvquine.org/


PHILOSOPHICAL
POWERS
WILLARD VAN ORMAN
QUINE ™
*WITH WEB OF BELIEF®!
MAKES ENEMIES BELIEVE THEY'VE ALREADY BEEN BEATEN!
ENEMIES AT THE EDGES ARE EASILY DEFEATED, BUT NO ONE IS SAFE FROM BEING REVISED IN BATTLE!
© 2004 Ian Vandewalker

Python procedural version

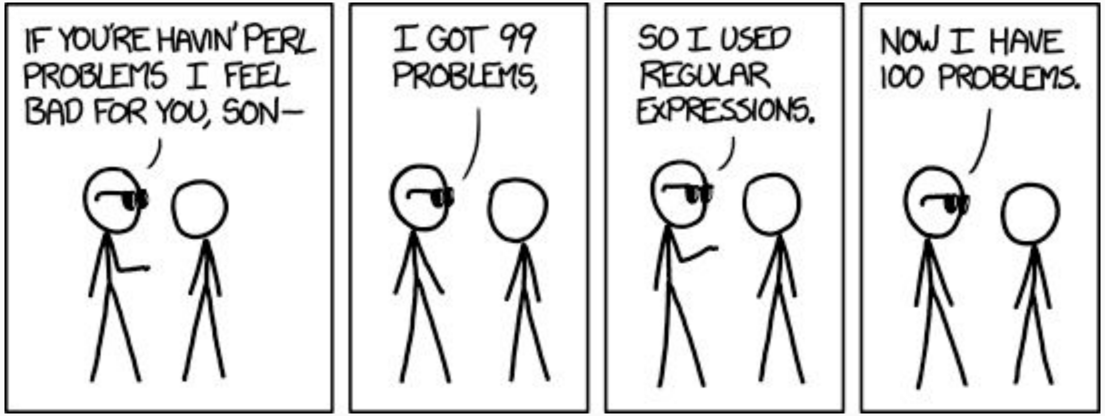
```
def procedure(input):
    output = input + 1
    return output


procedure(2)
```

Fortran (compiler)

```
PROGRAM MAIN

INTEGER N, X

EXTERNAL SUB1

COMMON /GLOBALS/ N

X = 0

PRINT *, 'Enter number of repeats'

READ (*,*) N

CALL SUB1(X,SUB1)

END


SUBROUTINE SUB1(X,DUMSUB)
```

Perl (compiler)

```
my $x = "foo";

my $some_condition = 1;

if ($some_condition) {

      my $y = "bar";

      print $x;          # prints "foo"

      print $y;          # prints "bar"

}

print $x;                # prints "foo"

print $y;                # prints nothing; $y has fallen out of scope
```


IF YOU'RE HAVIN' PERL PROBLEMS I FEEL BAD FOR YOU, SON—
I GOT 99 PROBLEMS,
SO I USED REGULAR EXPRESSIONS.
NOW I HAVE 100 PROBLEMS.

https://xkcd.com/1171/

Netlogo

```
to go
  if all? turtles [xcor >= food-x]
       [ stop ]
   ask leaders                                        ;; the leader ant wiggles and moves
       [ wiggle leader-wiggle-angle
       correct-path
       if (xcor > (food-x - 5 ))                      ;; leader heads straight for food, if it is close
       [ facexy food-x food-y ]
       if xcor < food-x                               ;; do nothing if you're at or past the food
       [ fd 0.5 ] ]
   ask followers
```

C (compiler)

```
#include <stdio.h>

int main() {

    int num;

    printf("Enter an integer: ");

    scanf("%d", &num);


    // True if num is perfectly divisible by 2

    if(num % 2 == 0)

        printf("%d is even.", num);

    else

        printf("%d is odd.", num);
```

PHP (compiler) - comics, dummies

```
<?php

// PHP code to check whether the number

// is Even or Odd in Normal way

function check($number){

       if($number % 2 == 0){

       echo "Even";

       }

       else{

       echo "Odd";

       }

}
```

AWK (compiler)

```
awk 'BEGIN { print "Hello, world" }'
```

Brainfuck (compiler)

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>.
```